REPRINTED FROM:

PERFORMANCE EVALUATION OF SUPERCOMPUTERS

Edited by

Joanne L. MARTIN
IBM Corporation
T. J. Watson Research Center
and
Data Systems Division
Yorktown Heights, NY 10598
U.S.A.

1988

NORTH-HOLLAND
AMSTERDAM • NEW YORK • OXFORD • TOKYO

Performance Evaluation of Supercomputers
J.L. Martin (editor)

MEASUREMENT AND ANALYSIS OF MEMORY CONFLICTS ON VECTOR MULTIPROCESSORS

Donald A. Calahan*

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109

David H. Bailey

NAS Projects Office
NASA Ames Research Center
Moffett Field, California 94035

ABSTRACT

The memory organization and technological design parameters which create memory access conflicts and affect performance of the CRAY family of processors are studied. Measurements on the dynamic-memory CRAY-2 system are presented.

1. INTRODUCTION

The literature of the 1960's and 1970's appeared to solve the multiprocessor (MP) memory conflict problem with the design of conflict-free memories. The real evolutionary world of scientific computers in the last decade, however, made only modest use of this theory, for the following reasons.

1. The fast clock periods of the early scientific multiprocessors mandated against use of extensive conflict resolution hardware in favor of simpler designs. It was deemed better to let conflicts occur, detect their occurrence, and then resolve them heuristically on-the-fly.

2. The low-parallelism (2 or 4) did not match the previous theoretical models nor require their intricate solution.

3. Initial conflict-free models dealt only with regular (vector) accessing, whereas early scientific processors were required to prove their worth with scalar accesses.

---

* This investigator was supported by the NAS Projects Office and the Air Force Office of Scientific Research under Grant AF 840096.

4. The use of slow massive dynamic memories as with the CRAY-2 (abbr. C-2) introduced a new - and sometimes dominant - source of delay, namely, the chip cycle time (bank reservation time). Thus, it would be unimportant that an access reached a chip through a clever routing if the chip were still recovering from the last access and so unavailable. This problem was exacerbated by the technology push for faster cycle time, since all events are denominated in clock periods (cps).

This chapter therefore does not speak to the avoidance of conflicts in a massively-parallel system (see [1]) but rather to the techniques of measurement and analysis related to conflicts in the CRAY family of multiprocessors.

2. MOTIVATION: CRAY-2 BENCHMARK STUDIES

Although the CRAY X-MP (abbr. X-MP) memory conflict resolution system poses some interesting theoretical design problems (see below), the overall machine performance is only marginally affected by conflicts. In contrast, the C-2 appears to suffer considerable performance loss because of memory conflicts. This is chiefly due to relatively slow dynamic RAM chips the C-2 employs .

Some examples of this slowdown on some actual Fortran test codes are shown in Table 1.

Table 1. Examples of interprocessor contention on the C-2.

Program Name	One-Processor Stand-Alone	Four-Processor Simultaneous	One-Proc. Normal	Percent Reduction
ARC3	47.72	33.67	35.04	26.6
BL3D	46.00	42.12	40.33	12.3
F3D	33.06	27.02	27.31	17.4
INS3D	59.74	47.44	45.68	23.5
LES	93.95	66.40	60.59	35.5
MATEST	404.02	278.30	279.69	30.8
NASKERN2	98.86	63.66	66.43	32.8
PITEST	167.13	163.93	154.28	7.7

The column headed One-Processor Stand-Alone gives the performance rate of a program in millions of floating-point operations per second (MFLOPS) when run on one processor with the

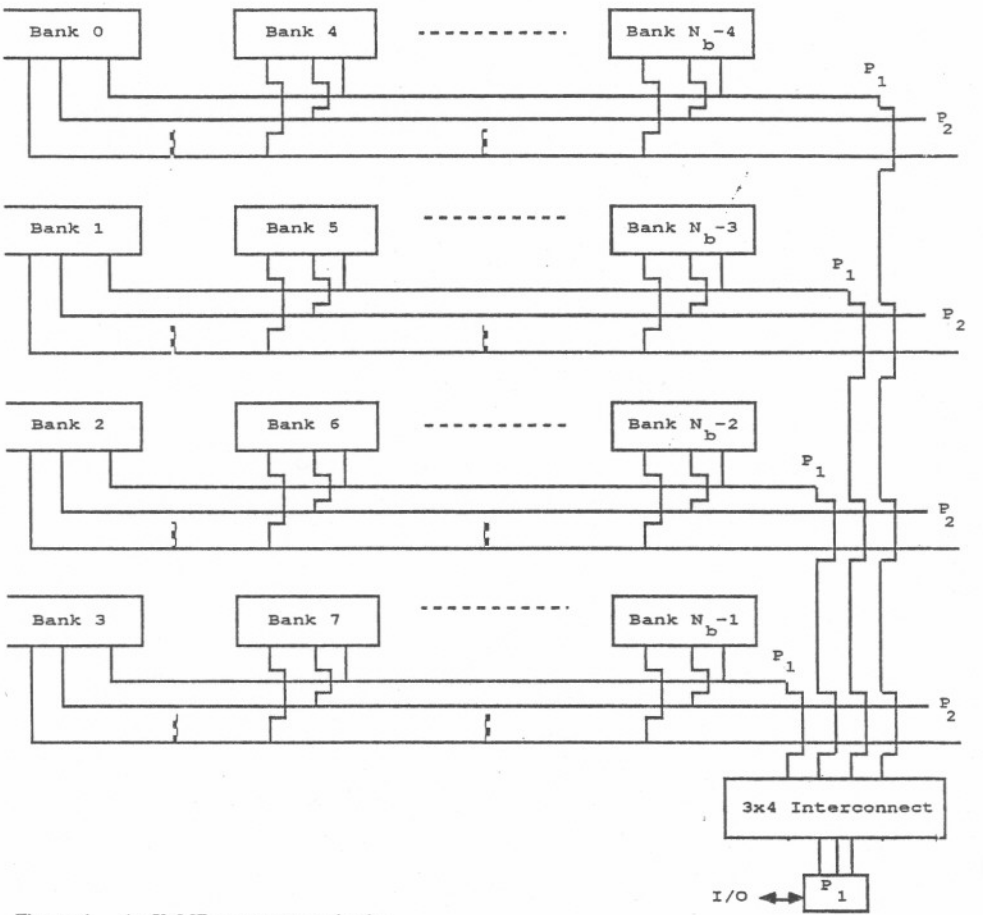

Figure 1. An X-MP memory organization

The critical design issue is the periodic manner in which the sections are connected to the banks. Specifically, if the number of sections is NS and the number of banks associated with the same section is NBPS, then X-MP bank number BN is connected to section number SN = mod (BN/NBPS, NS). Figure 1 depicts the connection for NS=4 and NBPS=1.

The design criteria of reducing access conflicts is represented in the literature by studies of *steady-state* vector access delay [2, 3, 4, 5] and *startup* vector access delay [6, 7, 8]. Both analyses assume that ties between conflicting accesses are resolved at the section level by a priority that is usually determined by the time of vector access instruction issue, so that an ongoing vector access has priority.

3.2.2 Steady-state Delay

An important class of steady-state vector conflicts is the 2-port *linked conflict* shown in Figure 2, where the section conflict resonates with the bank reservation time (=4 cps) to produce a 25% decrease in accessing rate. As described in [2], at cp=1 ports A and B place their initial requests for banks 0 and 3, respectively. Both are granted, so section 0 is reserved by A for 1 cp, and section 3 is reserved for port B. In addition, bank 0 is reserved by A and bank 3 is reserved for by port B; each for four cps. During cp=2 and cp=3, there are no conflicts. At cp=4, port A requests bank 3, which is still reserved for port B, so that port A must wait. At cp=5, bank 3 becomes available so port A gets it together with section 3 because port A has higher priority. At cp=5, port B needs bank 7 and section 3, but it must wait one cp for the section. Because of this delay, port B has bank 7 when A requests it at cp=9, and the cycle continues. As the figure shows, a steady-state delay of 1 cp is incurred every four cps.

In [2, 3, 4, and 5], other forms of steady-state delays are studied. It is shown that the above example occurs because the number of sections is equal to the bank reservation time (4 cps), and the conditions for avoiding such resonance are given.

Simulation studies in these reference using random accessing have shown the net effect on average access delay of this and similar steady-state delays [2].

3.2.3 Transient Response: Vector Startup Delays

The startup behavior of the conflict process is important for two reasons.

1. The number of vector startups is inversely related to the maximum vector length, which is short (=64) for the CRAYs.

2. Collisions necessarily result in restarts, producing startup-like delays.

In this analysis, an infinite-bank memory will be assumed, so that the probability of accesses in neighboring banks from the same processor (as in Figure 2) will be zero. Delays will be due solely to coupling between the periodic section numbering. The relationship between the memory organization (NS and NBPS) and the average startup delay may be determined by enumeration; i.e., all possible relative locations between ongoing accesses and an initiating access are considered equally likely and each produces a distinctive delay in the initiating access. These startups are

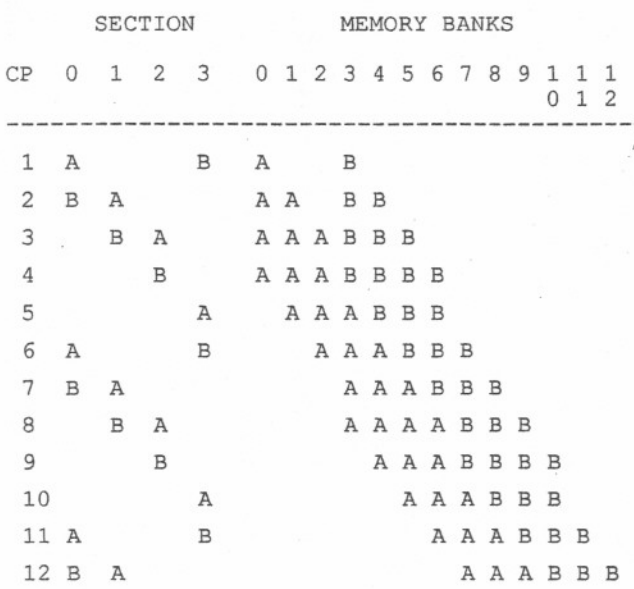

Figure 2. Reservation table for a linked conflict [2].

summed and an average startup delay determined. Enumeration is feasible due to the periodicity of the section numbering.

The results are shown in Table 2, depending on whether two or three ports are active; e.g., a three-port access assumes two ports are in a conflict-free accessing mode when a third access is initiated. In either case, it is shown that the original X-MP design (X-MP 24), with NBPS=1 and NS=4, produces significantly less startup delay than the later design (X-MP 48), with NBPS=4 and NS=4. A similar vector restart analysis produces a result which also favors the X-MP 24 design.

From the above analyses, it is clear that steady-state and transient characteristics are improved by different memory designs. Relative to future X-MP designs, it may be argued that, with the vector length fixed at 64 and with the number of banks increasing as more processors are added, the opportunities for neighboring linked conflicts from the same processor decrease; in contrast, startup conflicts result from a periodic section numbering, and are thus largely by changing but a large but finite number of banks. For this reason, it is conjectured that startup phenomena will be the more important consideration in future X-MP designs. Timing simulations involving complete programs appear to corroborate this contention [6, 7].

Table 2. Startup section delays as function of the number of sections (NS) and the number of banks per section (NBPS).

STARTUP DELAYS

NBPS	NS	2-port access (clocks)	3-port access (clocks)
1	2	.5	---
	4	.25	.67 (X-MP 24)
	8	.125	.28
2	2	1.5	---
	4	.75	4.2
	8	.38	.86
4	2	3.5	---
	4	1.75	5.11 (X-MP 48)
	8	.88	2.04
8	2	7.5	---
	4	3.75	11.2
	8	1.88	4.4
	16	.94	1.93
16	2	15.5	---
	4	7.75	23.5
	8	3.87	9.11
	16	1.93	4.15

3.3 Effects of Technology: The C-2

At the bank level, conflict resolution of a multiprocessor vector computer system may be approximately modeled using a relatively simple Markov chain model, relating only the number of banks and the bank reservation time. While such a model cannot precisely describe the phenomenon of memory bank contention in a real vector computer, it does serve as a good introduction to the problem, and in fact some quantitative conclusions can be drawn from this simple model that do carry over to more realistic models.

In order to facilitate analysis, certain simplifying assumptions will be made. It will be assumed that the computer system being modeled has m CPUs and n banks of interleaved memory (i.e., successive data words are in successive memory banks). It will be assumed that the cycle time for a complete memory access is t CPU ticks. In particular, it will be assumed that whenever one of the CPUs initiates an access to a word of memory (either to store or recall), a reservation of t ticks is placed on the bank containing that word. This means that for the next t system ticks, any CPU wishing to initiate an access to a word in that bank of memory must wait before it may begin. Once a CPU has initiated a memory fetch or store, it is free to initiate another at the next CPU clock period. Note that a single CPU may be simultaneously in the process of accessing up to t separate memory banks, provided no bank busy conflicts are encountered.

At each system clock tick, it will be assumed that each CPU that is not waiting tosses a coin with probability of heads equal to q, and attempts to initiate a memory access (from a memory bank chosen at random) if the coin turns up heads. It will be assumed that when a CPU attempts to access to a bank that is busy from a prior reservation, the remaining reservation on that bank is uniformly distributed between 1 and t. The case where more than one CPU is waiting to access a single reserved bank will be ignored in this Markov model. A final approximating assumption is that the fraction of memory banks that are busy at any time is approximately a constant x. Such an assumption may be made assuming that the process has achieved a steady state.

It should be mentioned that in real vector computer operation, a CPU is typically either attempting to access memory cells every tick, as part of a long vector fetch or store, or else "crunching" and not attempting to access memory at all. Further, most memory accesses are from consecutive memory banks, instead of from randomly chosen memory banks. This last deviation appears to be the most serious in the model. By comparison, the assumption that no more than one CPU is queued waiting to access a single busy bank does not appear to be a serious limitation, based on the results of empirical simulations.

The operation of each CPU may now be approximately modeled by a Markov chain on the $t + 1$ states $s_0, s_1, s_2, \ldots, s_t$. Here s_0 denotes the free state and s_k denotes the state of waiting for a bank that has a reservation of k ticks remaining. Let T denote the Markov transition matrix for this model (i.e., T_{ij} is the probability that the next state is j, given that the current state is i). Then T may be written as

$$\begin{array}{c} 1 - qx \;\; qx/t \;\; qx/t \;\; \ldots \;\; qx/t \;\; qx/t \\ \begin{array}{cccccc} 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ . & . & . & \ldots & . & . \\ . & . & . & \ldots & . & . \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{array} \end{array}$$

It may easily be verified that the Markov chain described by this transition matrix is a regular (ergodic) process. This means that the a priori probability of any state is equal to the limiting frequency of appearance of that state (for almost every sample sequence). Let $p = (p_0, p_1, p_2, \ldots, p_t)$ denote the vector of a priori probabilities of the $t + 1$ states. These probabilities may be determined from the relationship $pT = p$. This equivalence yields the linear system of equations

$$p_0(1 - qx) + p_1 = p_0$$
$$p_0 qx/t + p_2 = p_1$$

$$p_0qx/t + p_3 = p_2$$

.

.

.

$$p_0qx/t + p_t = p_{t-1}$$

$$p_0qx/t = p_t$$

When combined with the fact that the probabilities p_k must sum to one, the solution is easily found to be

$$p_0 = 1/[1 + qx(t + 1)/2]$$

$$p_1 = qx/[1 + qx(t + 1)/2]$$

$$p_2 = qx(t\text{-}1)/[t(1 + qx(t + 1)/2)]$$

.

.

.

$$p_{t-1} = 2qx/[t(1 + qx(t + 1)/2)]$$

$$p_t = qx/[t(1 + qx(t + 1)/2)]$$

Since it was assumed that the fraction x of banks that are in a reservation cycle is constant, the expected number of banks initially reserved at any instant must equal the number whose reservation expires at that instant. This can be expressed by the relation $q \times m \times p_0 = \frac{n \times x}{t}$ where it is assumed that at each time 1/t of the busy banks are freed. This relation combined with the above yields the solution

$$x = \frac{\sqrt{(1 + 2mq^2t\frac{(t+1)}{n})} - 1}{q(t+1)}$$

so that

$$p_0 = \frac{2}{1 + \sqrt{(1 + 2mq^2t\frac{(t+1)}{n})}}$$

The remaining p_k can be similarly calculated.

Now that the probability vector p has been found, a memory efficiency statistic may be calculated. Let E denote the ratio of the expected number of memory accesses divided by the sum of this figure and the expected number of CPU ticks spent in wait states. This efficiency statistic can be written as

$$E = q\frac{p_0}{qp_0 + 1 - p_0}$$

$$= \frac{2q}{2q - 1 + \sqrt{(1 + 2mq^2t\frac{(t+1)}{n})}}$$

This formula for the efficiency statistic does not, unfortunately, agree closely with most actual vector supercomputer operation. The main problem appears to be, as mentioned above, that most vector computer memory accesses are from consecutive banks (or at least from banks differing by some constant stride) instead of from randomly chosen banks. The term stride here refers to the increment in memory between successive elements in a vector fetch or store. Only in the case where a computer is running programs with uncorrelated nonunit strides does this formula closely agree with actual memory performance.

In spite of these limitations, the above formula does contain implicit relationships between the number of processors, the number of banks, and the bank reservation time that do carry over, to more realistic models. First of all, one can conclude from this formula that if the number of processors m is increased by a factor k, then the number of banks n must also be increased by a factor k to preserve the same level of efficiency. Secondly, if the bank reservation time t is increased by a factor k, then the number of banks must be increased by a factor of about k_2 to maintain the same memory efficiency.

Sophisticated models can be formulated that more accurately model a real multiprocessor vector computer system. One such advanced model will now be presented and briefly studied. For complete details, see [10].

Above it was assumed that each free CPU tosses a coin with a certain probability and attempts to access a single randomly chosen memory bank if the coin turns up heads. In the following it will be assumed that each free CPU instead initiates a vector access (fetch or store) of a certain length if its coin turns up heads (which occurs with probability r). The starting bank number for this vector access is assumed chosen at random, but thereafter the bank number advances with some constant stride through the duration of the vector access. The length of the vector access is assumed chosen at random according to a distribution that is uniform on the set {1, 2, ..., V}, except that a specified larger fraction V of the vector lengths have the maximum value V. Similarly, the memory stride is assumed to be chosen from a uniform distribution on the set {1, 2, ..., n}, except that a certain specified larger fraction s of the strides are 1. As in the Markov chain model, it will be assumed that a reservation of t ticks is placed on any memory bank once a CPU initiates a memory access. Unlike the Markov chain model, this model will not ignore the case where two or more CPUs are waiting to

access the same memory bank -- it will be assumed that the CPUs merely take turns until all accesses have been completed. Observe that if no conflicts are encountered, a single CPU can be simultaneously accessing up to t separate memory banks.

Unfortunately, it does not appear to obtain analytic solutions using Markov techniques as above for such a model. Such models can, however, be investigated using Monte-Carlo simulation methods. Various assumptions of the above parameters were simulated for one million CPU ticks, enough to insure that empirical statistics are accurate to within one or two percent.

Several plots displaying important simulation results are shown on the next page. Except where indicated otherwise, these results are for the case n = 256, m = 4, V = 128, R = 1/r = 100, t = 40, v = 0.75, s = 0.75. These parameters were chosen for a "generic" vector computer, roughly a composite of a number of current and projected supercomputers.

Figure 3 shows how the memory efficiency E decreases as the reservation time t increases. The four separate curves represent results for various numbers of CPUs. Figure 4 shows how efficiency increases as the fraction s of unit stride varies from zero to one. Each curve in this figure represents results for different reservation times. Figure 5 shows how efficiency decreases with large numbers of processors. The four curves on this figure are for different numbers of banks. Figures 6 and 7 present a different slant on the problem: with other parameters held fixed, the number of banks necessary to preserve a constant level of memory efficiency (75%) is shown as a function of increasing reservation time (Figure 6) and as a function of increasing numbers of processors (Figure 7). In Figure 6 the separate curves represent results for different numbers of banks, and in Figure 7 each curve gives results for different reservation times.

Several definite trends can be quickly identified from these plots. First of all, from Figure 7 it is clear that the relationship between banks and processors is exceedingly close to linear. To be precise, the number of banks necessary to compensate for an increasing number of processors appears to be very closely proportional to the number of processors minus 1. This relation, except for the minus 1, matches the relation found in the Markov chain analysis above. Secondly, although it is not immediately clear from Figure 6, logarithmic regression of the simulation results shows that the number of banks necessary to compensate for an increase in the bank reservation time t is proportional to approximately $t_{1.85}$. The corresponding relation from the Markov chain analysis is $t(t + 1)$, which is equivalent to approximately $t_{1.96}$ over the range of the data in question. Relationships quite close to these were also found in other cases that were run with the simulator program.

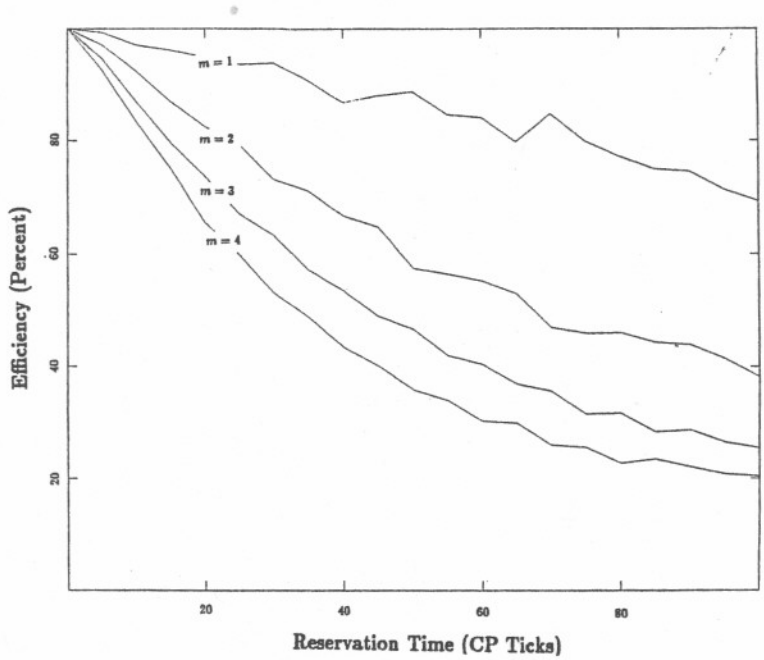

Figure 3. Efficiency relative to reservation time.

The near-linear relationship between the number of processors and the number of memory banks needed to preserve a tolerable level of memory contention has been assumed for some time. However, the near-quadratic relationship between memory bank reservation time (measured in clock periods) and the number of memory banks is somewhat surprising. At present, only the C-2 presents an example of a system with a large enough memory bank reservation time that these modeled results can be compared with real systems, and the severe contention on the C-2 does appear to be roughly in accordance with these projections. Whether or not this trend will be upheld in other systems remains to be seen. In any event, these results underscore the potential for truly catastrophic performance reductions if memory bank contention is not carefully considered in the design of a supercomputer system.

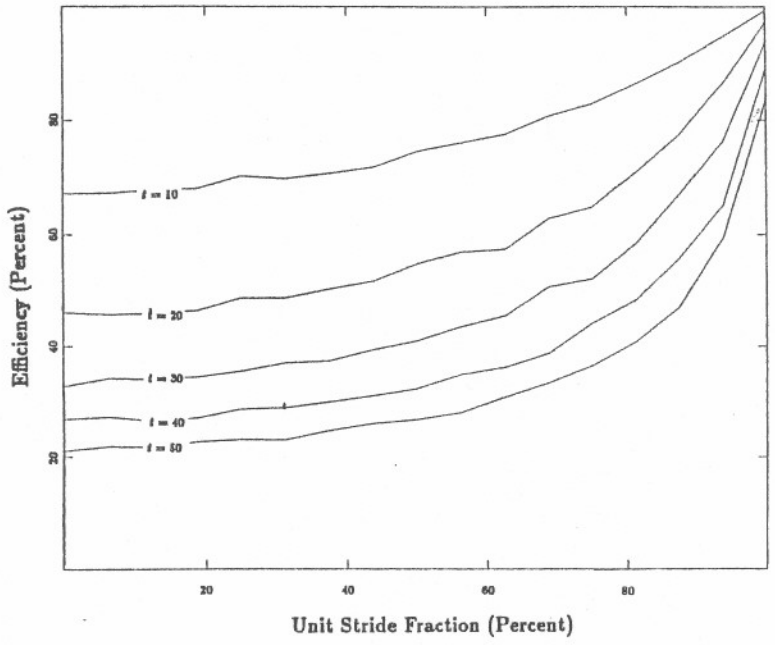

Figure 4. Efficiency relative to unit stride fraction.

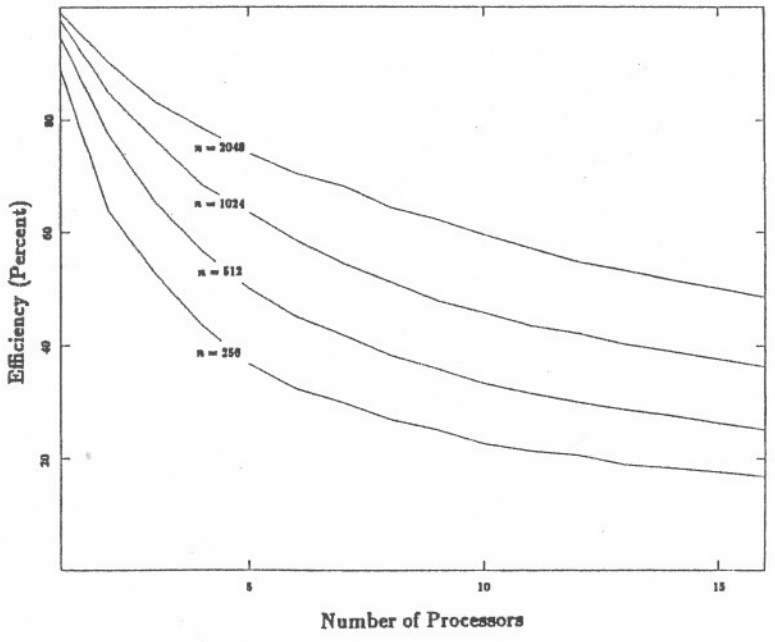

Figure 5. Efficiency relative to number of processors.

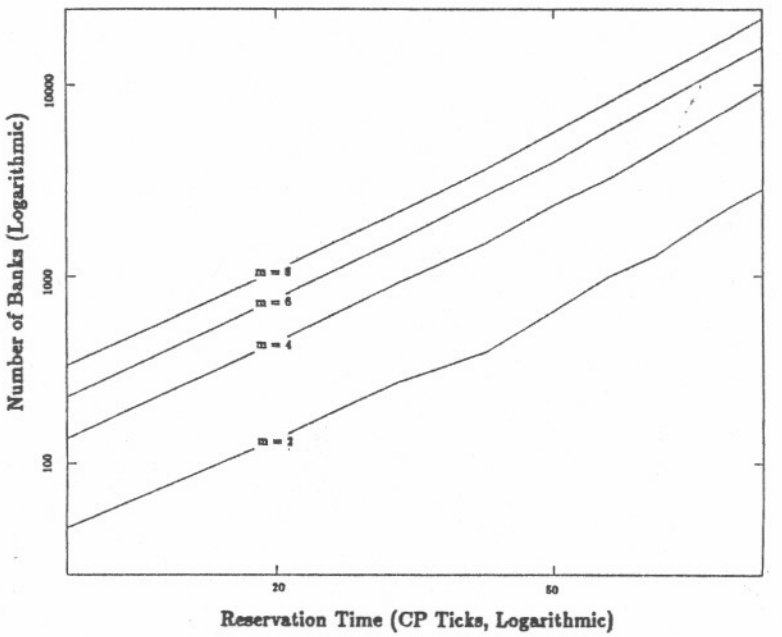

Figure 6. Number of banks relative to reservation time.

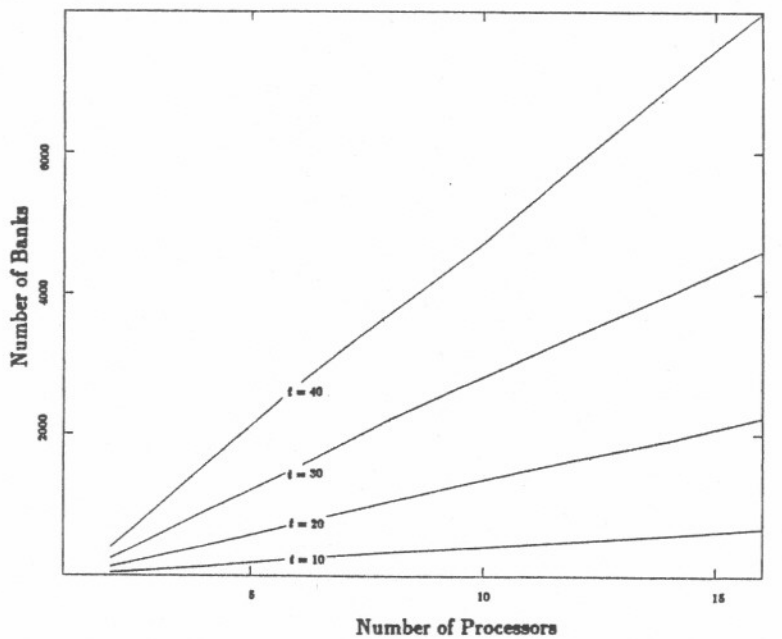

Figure 7. Number of banks relative to number of processors.

4. EVALUATION: PERFORMANCE CHARACTERIZATION AND MEASUREMENTS

4.1 Introduction

The above analysis speaks primarily to the memory system design. However, these are not the only issues that determine conflict-related performance degradation of algorithms in a real system. Most of the following issues have some relevance to scalar MP processing; all appear far more important in vector multiprocessors.

1. Algorithm sensitivity. The delay observed in an algorithm is only partly attributable to the delay in the memory system itself. The manner in which the algorithm reacts to such delays is equally important. From this observation, useful measurement probes can be constructed which have generic value across classes of vector algorithms.

2. Performance variability. Variability of performance due to load variations in a production environment is an issue which should be measured and evaluated to put any other set of measurements in context.

3. Load modeling. Heretofore undistinguished properties of the memory loading which an algorithm encounters can significantly alter its degradation.

4. Other system attributes. The operating system and other hardware features can have an effect which deserves evaluation.

These topics will be considered in the following sections, although not necessarily delineated in the same manner.

4.2 Access and Algorithm Delays [9]

4.2.1 Definitions

Let T_{0i} be the clock period that the ith memory vector access instruction reserves a memory port, and let T_{1i} be the first cp that the memory port is free for a later access. Define the memory access time as $T_{aci} = T_{1i} - T_{0i}$. Let the conflict-free access time be T_{acfi}. For a sequence of N accesses, define the average access delay

$$D_{\mathrm{ac}} = \left(\frac{1}{N}\right)\sum_{i=1}^{N}(T_{aci} - T_{acfi})$$

and the average per cent access delay as

$$D_{ac\%} = \left(\frac{100}{N}\right) \sum_{i=1}^{N} \frac{(T_{aci} - T_{acfi})}{T_{acfi}}$$

Similarly, let T_{ali} and T_{alfi} be the measured and the conflict-free times for an algorithm to execute. For a sequence of N algorithm executions, define the average algorithm delay

$$D_{al} = \left(\frac{1}{N}\right) \sum_{i=1}^{N} (T_{ali} - T_{alfi}) \tag{1}$$

and the average per cent algorithm delay as

$$D_{al\%} = \left(\frac{100}{N}\right) \sum_{i=1}^{N} \frac{(T_{ali} - T_{alfi})}{T_{alfi}}$$

4.2.2 Sensitivity and measurement probes

Consider the ith execution of an algorithm of M vector accesses, so that the total delay is $D_t = MD_{ac}$. Then define the algorithm sensitivity of the ith execution to access delay as $S_{al} = \frac{D_{al}}{D_t}$

This sensitivity has several properties.

1. Without chaining, $S_{al} \leq 1$. This follows from the fact that the effect of a memory access delay on D_{al} cannot be magnified by subsequent events; however, its effect can be diminished by algorithm insensitivity to access delays. Indeed, algorithm coding can often be devised to reduce S_{al} by, for example, the prefetching of operands or their prestorage in conflict-free local memories.

2. Of all possible algorithms with M accesses delayed by total time MD_{ac}, the maximum D_{al} is incurred by an algorithm of vector accesses only, with $S_{al} = 1$. Such a vector code (designated VECTOR READ or VR), with successive 64-length unit-stride accesses initiated from banks 64 apart and instrumented to record all delays, will be a sensitive probe used throughout this research. When maximum-rate accesses are made so as to constantly busy an access path, virtually any shared-memory activity will create some delay. Note that this does not require that a specific execution of such a code with unspecified memory loading result in the largest $D_{al\%}$,

since D_{ac} depends on accessing patterns and attributes relative to the other memory acccsses. For example, vector accesses with negative strides are notorious for creating large delays in a positive-stride load environment; it is possible that fewer such accesses could produce more delay than does VR.

Simulation shows [9] that typical code sensitivity is only mildly dependent on the memory loading, so that using $D_{ac} = S_{al}D_r$, the sensitivity can be considered a constant code attribute that maps access delays - which are architectural design objectives - into the algorithm delays seen by the user. Unfortunately, any software probes to measure S_{al} in a given code - except by simulation - risk disturbing its value. Also, simulation has shown [7] that it can vary widely between codes. Thus, the concept has value in determining an upper limit in the level of certainty we can have in predicting or explaining observed algorithm delays without detailed knowledge of the low-level code organization.

4.3 Dynamic Analysis

Dynamic profiling with the VR test is quite useful in delineating the impact on delay of certain architectural and run-time characteristics. In this test, both (1) the individual delays of consecutive reads and (2) a running average (D_{rave}) - with a averaging period equal to the memory refresh period - will be displayed as a function of time.

Delay from a number of sources have been distinguished with such tests applied to the C-2.

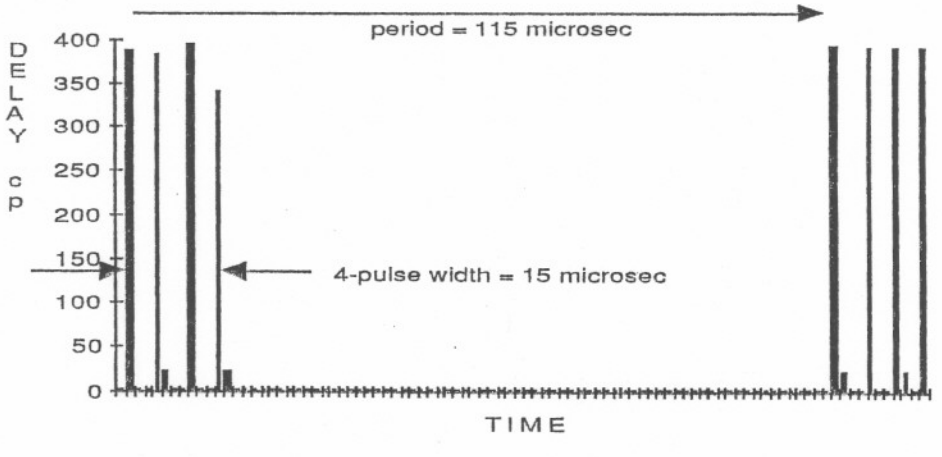

Figure 8. Effects of refresh on VR test. Measurements made on MFECC C-2 in 8/85.

1. From memory refresh. This accounts for periodic long delays in vector accesses at regular intervals. Figure 8 indicates delays of nearly 400 cps in four successive vector reads at the refresh rate of 115 cp; these delays, measured in a quiet background, are also quite distinctive with a fully-loaded machine. The time-average effect of these massive delays (D_{rave}) is 3.6 cps, so that refresh is shown to have negligible overall impact.

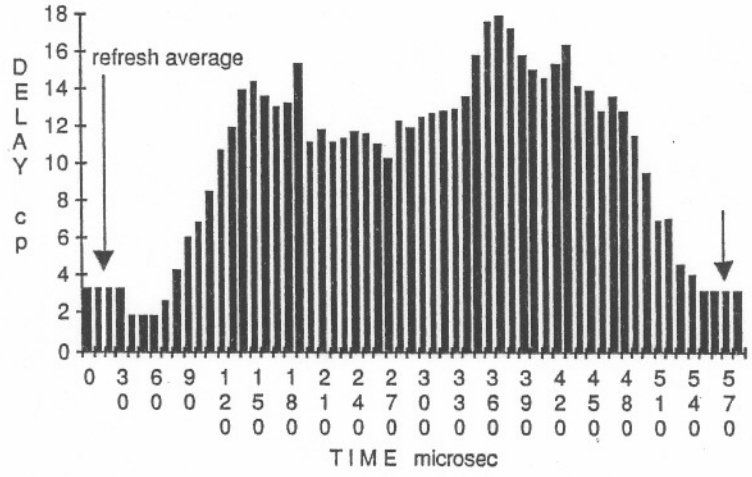

Figure 9. Effects of operating system on VR test. Measurements made on MFECC C-2 in 8-85.

2. From the operating system. In the UNICOS operating system, the operating system usually visits only a single processor, whereas the CTSS operating systems uses a roundrobin algorithm. Figure 9 depicts D_{rave} during such a visitation during a dedicated run on the MFECC C-2; the average delay is observed to be in the range of 15 cp, against the above-mentioned 3.6 cp background. This delay can of course vary; the suspicion is that it will increase markedly when I/O is being performed.

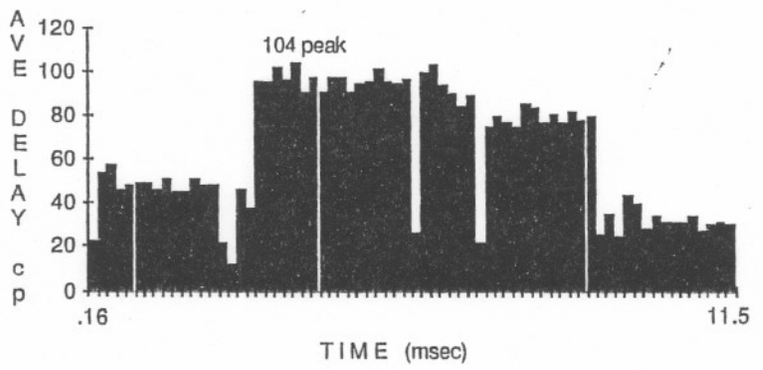

Figure 10. Effects daytime load on VR test. Measurements made on MFECC C-2 in 6-86.

3. From user codes. Figure 10 shows the running average delay taken during a daytime load at MFECC. The extent of variation of the average is quite remarkable, with sustained delays of 45, 100, and 35 cp over three regions in only 11 ms! Clearly, kernel and other short timings made against this background will vary significantly, a common user observation on the C-2. It is conjectured from simulation experience that this wide variation is caused by a nonlinear response of the delay to the amount of memory traffic, so that small additional traffic causes a type of avalanche or seizure effect from which memory is slow to recover; this could in turn result from resonances in the quite complicated C-2 memory design.

4.4 Static Analysis: Delay Distribution Functions

The impact of other architectural features can be delineated by observing the number of delays recorded at each value of delay, or the delay distribution function (DDF [9]). The effect of internal buffers in the C-2 memory system are evident in Figure 11, where only a discrete spectrum of delays is observed even in the mixed load of a daytime load environment. In Figure 12, the X-MP shows a continuous spectrum of delays; however, the effect of 11-cp instruction buffer fetches from another user - which can interrupt ongoing vector accesses - causes an increased number of delays at 11-cp intervals in the distribution. These peaks continue at 11-cp intervals, representing vector accesses that encounter 2 or more buffer fetches.

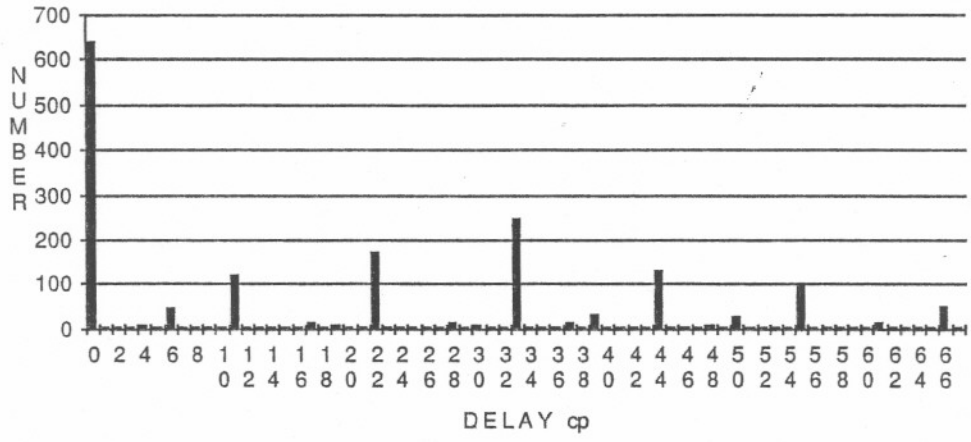

Figure 11. Delay Distribution Function during daytime load on C-2. Measurements made on NAS C-2 in 6/87.

4.5 Effects of Scalar Accesses

Intuition predicted and simulation has verified that delay in a vector access would be a function of both the number and regularity of conflicting accesses; e.g., a 64-length vector access would be a less disruptive load than 64 randomly addressed scalar accesses. Nonetheless,the large D_{rave} produced by the operating system in Figure 9 was felt to be uncharacteristically large, since it was produced by a single processor executing system software that was known to be largely scalar and so would produce modest memory traffic. Seemingly, three processors running similar scalar software could produce 45-cp access delays in a test code!

To investigate the effects of scalar accesses, a series of tests were devised to include exclusively scalar and exclusively vector loads and test codes (four test series). Figure 13 shows the model adopted in this series. Processors P2,P3, and P4 each execute either (1) an identical LOAD CODE with known access characteristics or (2) an idling code with no memory accesses; these are intended to provide memory traffic for an instrumented TEST CODE run in P1, for which access delays were measured.

Partial results are shown in Table 3, when P2,P3, and P4 each ran identical codes; the incremental effects of idling any load processor(s) appeared to have a linear effect on delay and will not be displayed. Here, the SORT LOAD CODE produces one scalar access approximately once every 50

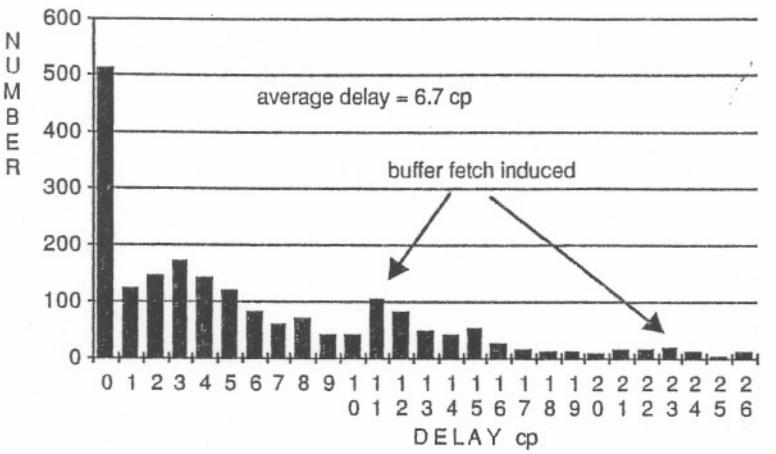

Figure 12. Delay Distribution Function during daytime load on X-MP. Measurements made on MFECC X-MP-24 in 6/87.

cps, whereas the M*M matrix multiply LOAD CODE produces nearly continual 64-length vector accesses.

It is remarkable that, whereas the scalar TEST CODES are affected by the large differences in the absolute number of memory accesses of the two LOAD CODES, the vector TEST CODES are degraded equally by the light scalar and the intense vector loads. This corroborates the observation of significant operating system loading on the VR test code in Figure 9.

It is clear that a memory system performance which is so severely degraded by scalar accesses is quite undesireable. These results have suggested the following research.

A different LOAD CODE which produces prescribed no-load vector or scalar accessing attributes is being prepared. A similar instrumented TEST CODE is in development. The intention is to collect quantitative delay information relating accessing rate and regularity of load and test codes. It is expected that various nonlinear effects will be observed which can be identified with memory technology and design characteristics. The ultimate goal is to develop standardized test sequences that can be used in simulators at the development stage to predict performance of real codes for different memeory designs.

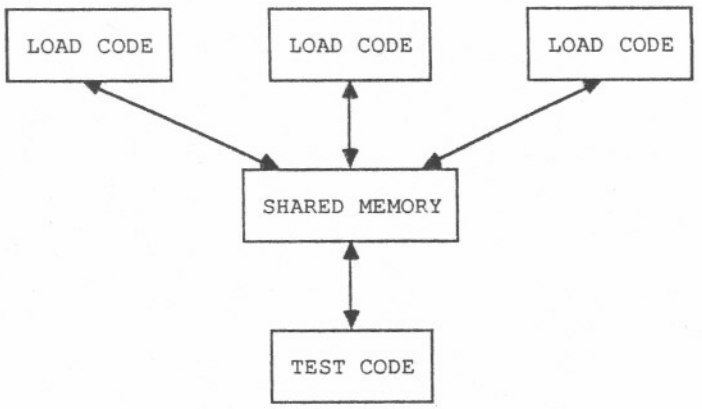

Figure 13. Experimental Model.

5. CONCLUSIONS

The CRAY-2 was designed as a research rather than commercial machine to test the two features of fast clock time and a massive common memory, relative to the state-of-the-art. It is therefore likely that future machines will not suffer from the present exaggerated memory chip cycle time that

	LOAD CODES	
	Scalar (SORT)	Vector (M*M)
TEST CODE		
Scalar		
GATHER	2.6%	18.3%
SORT	3.2%	26.3%
Vector		
FLUIDS KERN.	32.0%	34.7%
UNROLLED M*V	73.1%	85.6%

Table 3. Algorithm delay of test codes. All codes are Fortran. Run on NAS C-2 on 5/10/86.

accounts for the measured C-2 performance of this chapter. However, two issues are likely to keep this topic on the forefront of research.

1. At the design stage there will always be a tradeoff between memory speed and size, regardless of whether these are offered as product choices. Thus, both the manufacturer and the user may be well advised to develop enlightened testing procedures that predict the impact of conflicts on user codes.

2. It should be kept in mind that the memory-processor bandwidth of the C-2 is less than that of the X-MP, which has more memory ports. Consequently, a decrease in bank reservation time (in seconds) could easily be offset by an increase in memory bandwidth from the addition of more ports, from parallelism, or from a reduced clock period. Thus, an adventurous future design could restore the present criticality.

ACKNOWLEDGEMENTS

The programming assistance of students Geoffrey Carpenter of the University of Michigan is acknowledged. CRAY-2 time supplied by the Magnetic Fusion Energy Computing Center (MFECC) at Lawrence Livermore National Laboratory is appreciated.

REFERENCES

[1] Kuck, D.J., **The Structure of Computers and Computations** vol. 1,Wiley (1978).

[2] Cheung, T. and Smith, J.E., *An Analysis of the CRAY X-MP Memory System*, **Proc. 1984 Intl. Conf. on Parallel Processing**, Bellaire, MI, August 21-24, (1984) pp 494,505.

[3] Cheung, T. and Smith, J.E., *A Simulation Study of the CRAY X-MP Memory System*, **Trans. IEEE**, vol. C-35, (July 1986) pp. 613-622.

[4] Oed, W. and Lange, O. *On the Effective Bandwidth of Interleaved Memories in Vector Processing Systems*, **Trans. IEEE**, vol. C-34, (October 1985) pp. 949-957.

[5] Oed, W., and Lange, O. *Modelling, Measurement, and Simulation of Memory Interference in the CRAY X-MP*, **Parallel Computing**, vol 3, no. 4 (October 1986) pp. 343358.

[6] Calahan, D.A. *Memory Conflict Simulation of a Many processor CRAY Architecture. PART I: A CRAY X-MP Study*, Report SARL #6, EECS Dept., University of Michigan, (April, 1985).

[7] Calahan, D.A., *An Analysis and Simulation of the CRAY XMP Memory System*, **Proc. First Intl. Conf. on Supercomputing Systems**, St. Petersburg, FL, (December 1985) pp. 568-574.

[8] Calahan, D.A., *An Analysis of Vector Startup Access Delays*, **Trans. IEEE on Computers**, in **print.**

[9] Calahan, D.A., *Conflict Sensitivity of Algorithms. Part I: A CRAY X-MP Study*, Report SARL #7, EECS Dept., University of Michigan (April 1985).

[10] Bailey, D.H., *Vector Computer Memory Bank Contention*, **Trans. IEEE**, Vol. C-36, No. 3 (March 1987) pp. 293-298